THE ABC OF BBC

We continue our appraisal of the built-in BASIC of the most popular home computers by looking at BBC BASIC. This dialect is as full of fascinating abilities and facilities as the machine itself; and just as the BBC Micro marked a new phase in the design of home computers, so BBC BASIC is considered amongst the best of the dialects.

BETWEEN THE LINES

Don't forget to number your program lines in multiples of ten. Even the best programmers have to make insertions into their programs from time to time....

The criticism most often made of BASIC is that it is an unstructured language that encourages (or at least does nothing to check) bad programming habits in the beginner, especially the 'quick and dirty' approach to problem solving, which leads, for example, to undisciplined use of GOTO.

The use of ELSE with the IF..THEN statement can eliminate the commonest use of GOTO, by permitting both true and false cases of a condition to be dealt with in the same statement. For example, these lines:

```
1500 IF TEST >0 THEN GOTO 1800
1600 PRINT "VALUE OUT OF RANGE"
1700 GOTO 1900
1800 PRINT "NO PROBLEM"
1900 NEXT L
```

can be replaced by:

```
1500 IF TEST>0 THEN PRINT "NO PROBLEM"
     ELSE PRINT "VALUE OUT OF RANGE"
1900 NEXT L
```

GOSUB usually takes a line number as its argument, which has two disadvantages — first, GOSUB 1000, for example, gives no clue as to the purpose of the subroutine at line 1000; and second, specifying line numbers makes the program very difficult to renumber or merge.

GOSUB, like GOTO, is relatively slow in execution because the specified line has to be searched for in the program every time the instruction is obeyed.

BBC BASIC's functions and procedures answer these objections. Both are subroutine-like blocks of code, but are called by name rather than line number, so they can be self-documenting, or at least meaningful in the list, and need not be affected by subsequent renumbering or merging. Furthermore, function and procedure calls are generally executed more quickly than the GOSUB and GOTO commands.

Procedures and functions begin with DEF PROC or DEF FN, followed by a name, and usually (but not necessarily) a parameter list. For example:

```
1200 DEF FNcalc(a,b,c)=(a-b)*c/100 and
2500 DEF PROCoperate(w,xS,yS,z)
```

The definition will use these parameters as if they were program variables. When the program calls the function or procedure, however, the parameters, or dummy variables, may be replaced by any variable number or literal expression of the same data type as the original parameter. For example:

```
250 result=FNcalc (price,cost,12)or
545 PROCoperate (6, namesS, "smith",array(12))
```

The values of the parameters are then used in place of the dummy variables in the definition. Notice that a function can be used in an expression as if it were a variable or arithmetic quantity, whereas a procedure call must be used as if it were a BASIC command. The LOCAL command, which defines variables for use exclusively inside the definition block, removes the chance of a common subroutine bug. For example, consider this code:

```
100 FOR K=1 TO 10:GOSUB 500:NEXT K:END
500 FOR K=1 TO 5:PRINT"*****":NEXT:RETURN
```

Here the variable K is used as the loop counter in the main program line 100, and again in the subroutine in line 500 — an oversight that will seriously affect execution, but which can be extremely difficult to avoid (or to trace in a long program). In a BBC procedure, however, this danger is avoidable:

```
100 FOR K=1 TO 10:PROCstars:NEXT:END
500 DEF PROCstars
520 LOCAL K
540 FOR K=1 TO 5:PRINT"*****":NEXT
560 ENDPROC
```

The LOCAL command means that between lines 500 and 560 the variable K is a new variable, independent of the variable K anywhere else in the program, and having no effect on the value of K elsewhere. (Notice that PROCstars is a procedure without parameters.)

REPEAT..UNTIL is a loop structure in which iteration continues until the conditional expression that follows the word UNTIL is true; control then passes to the statement after UNTIL. For example:

```
200 DATA 12,234,31,45,65,0,76,81
250 REPEAT
300 READ number:sum=sum+number
350 UNTIL number=0
400 PRINT "Sum is ";sum
```